Proposal of a Logical Sensor Architecture using WoT-based Edge Microservices

Kazuki Miyagoshi[1], Yuuichi Teranishi[2,1], Tomoya Kawakami[3,1], Tomoki Yoshihisa[1], Shinji Shimojo[1]

1. Osaka University, Japan,
2. National Institute of Information and Communications Technology, Japan,
3. Fukui University, Japan

***Abstract*—This paper proposes "WoT-based Logical Sensor Architecture (WLSA)," a novel virtual sensor architecture on the basis of Web of Thing (WoT) that enables IoT applications to treat the sensor data and processing results in a uniform way in edge computing environments. WLSA reduces the computing and network resources required for the IoT applications by reusing the processing results in multiple data flows. This paper also proposes a data flow transforming (DFT) algorithm, in which a subset of the data flow is automatically replaced with a process that reuses the result of running processes so that the application developers do not need to conscious of sharing the processing result. We have implemented a prototype system and an object detection application based on WLSA using Node-RED data flow processing framework in an edge computing environment and evaluated its effectiveness. The evaluation results showed that WLSA reduced the execution time with less usage of computing resources. The prototype could accommodate 20 applications with less than 100 ms response time under 4% CPU usage.**

***Index Terms*—Web of Things, Edge Computing, Microservices**

1. Introduction

In recent years, the number of IoT sensor devices that can acquire data over the Internet has been growing rapidly. Collecting and analyzing real-time data from IoT devices on the cloud has led to more active efforts to improve the operational efficiency and functionality to automate their operations. In particular, applications for processing stream data, that requires continuous and real-time data processing, have been rapidly increasing. A typical example application is automatic control of air-conditioning appliances according to the temperature, humidity, and number and position of people in the room observed by sensors and cameras. We assume such automatic appliance control applications as a typical IoT service. Especially, we treat an application that improves air conditioning functions by processing video data from cameras.

In such stream processing applications that combine different functions, there is a problem of system isolation (siloization), which cannot inter-connect different functions due to the difference of the protocols, data definitions, etc. To cope with such a siloization problem, a concept called WoT (Web of Things) [1], have been proposed in which sensors and sensor data are provided by the standardized Web technology.

There have been many research and developments of the system called a service mesh, which defines

a data flow by connecting microservices in a cloud to implement stream data processing applications. To the ease of development and the ease of using existing assets, the service mesh often uses Web technology (e.g., HTTP/HTTPS) to connect different microservices. There have been several GUI-based programming environments, such as the WebThings Gateway [2] and Node-RED [3], that allow the service developers to describe the service logics by connection relationships and processing logic between Web microservices.

In edge computing environments, which have been standardized by ETSI [4], [5], such cloud microservices are offloaded and executed on edge computers for effectiveness and short response time. However, because the edge computers are implemented by power-saving devices in general, there is a problem of the resource limitations that the amount of computing resources is small, and the execution time becomes long when the processing request increases. ISTIO [6], a network function that realizes a service mesh, has a function to share and distribute a single microservice among multiple IoT services. By using these distribution functions to share microservices on edge computers among multiple IoT services, the overhead of running multiple processes can be reduced. However, in this method, since data processing for each service is executed independently, the amount of resources to be used cannot be reduced in processing, requiring a large amount of calculation resources per single data such as image processing.

In order to solve these problems, this paper proposes a "WoT-based Logical Sensor Architecture (WLSA)," which can reduce the resources required for data processing by reusing the processing result. WLSA proposes a notion of WoT logical sensor for WoT-based microservices to reuse the processing results in multiple IoT applications.

The contributions of the paper are as follows:

- We present a novel logical sensor deployment architecture of WoT-based edge microservices called WLSA.
- We present a design and implementation of the architecture.
- We demonstrate the effectiveness of our architecture by evaluations using prototype implementation.

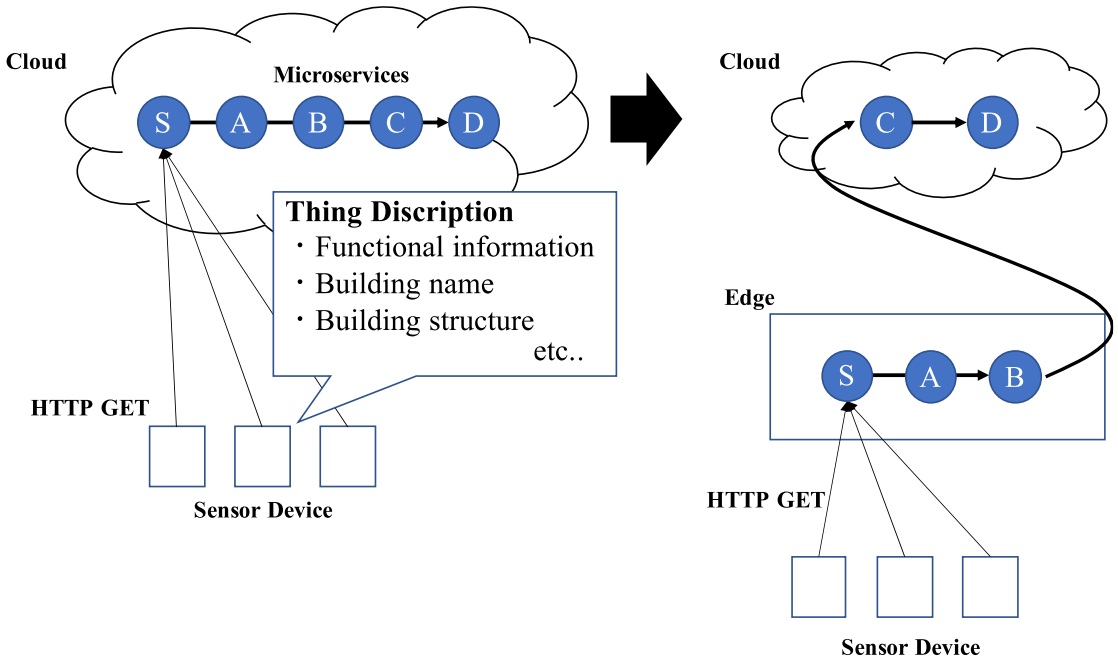

Figure 1. Edge microservice system model

2. Assumed Environment

Figure 1 shows the edge microservice system model we assume in this study. This paper assumes a WoT-based microservice architecture. The assumed environment is as follows.

- The sensor is managed by WoT. Each sensor has a URI and can provide access to data by HTTP/HTTPS.
- Microservices have URIs and can provide services by HTTP/HTTPS request-response.

WoT is a concept proposed by the W3C to avoid siloing in IoT and to utilize web technologies in IoT operations. The idea of the WoT is to create a decentralized Internet of Things by giving things URIs on the web to make them linkable and discoverable, and defining a standard data model and APIs to make them interoperable. WoT is intended as a unifying application layer for IoT, linking together multiple underlying IoT protocols using existing web technologies such as HTTP, Javascript, and REST. W3C standardized technologies are applied to sensor devices, and the sensor devices themselves are treated like the web, enabling interoperability between IoT applications. WoT gives each sensor a property in the form of a Thing Description (TD). In this study, it is assumed that not only the functional information of the sensor but also information such as the building name and structure are attached to the TD. In edge computing, these properties, data policy, network path, and execution efficiency are taken into account to determine the placement of the microservices to which the sensors are input. When the application defines a processing request as a data flow, the application executes the microservice deployed on the cloud or offloaded on the edge computers.

We assume that each data process in the data flow is stateless class [7]. The stateless class processes

are totally reentrant since it has no memory. Thus, the processes of this class can be used or reused in any context. We also assume the data flow is represented as a pipeline or combination of pipelines. Complex logical data flow structures can be expressed by connecting the input and output of multiple pipelines using the mechanism of many-to-many data delivery system such as publish/subscribe. Generally, the structure of the combination of multiple pipelines is represented as a directed acyclic graph (DAG). Such a pipeline data flow model with a stateless class can be applied to a wide variety of applications. Therefore, this model is adopted in the stream data processing in OSS platforms such as Spark [8], Beam [9], etc.

In the above-assumed environment, we define the requirement for the platform to execute WoT-based edge microservice as follows:

- Minimize the processing latency of data

  IoT applications assumed in the edge computing requires low latency. The process result needs to be obtained as quickly as possible to catch up with the dynamic situation changes. Some IoT applications request the processing result periodically. In this case, the platform should be able to return the response by the time the application issues the next request.

- Maximize the number of running applications

  We assume multiple applications run in the edge network. Though there is a limitation in the edge computing resources, the platform should accommodate as many applications as possible, keeping the process performance.

3. WLSA

3.1. Overview of WLSA

In this paper, we propose an architecture called WLSA that treats the output ofWoT-based microservices as a virtual WoT sensor (logic sensor) and makes it reusable.

WLSA assumes that an application is defined as a data flow, which runs multiple data processes to apply to the data generated by a sensor. WLSA also assumes that the data input is stream data, which is generated periodically and continuously from a sensor.

The basic idea of WLSA is as follows:

- The platform provides the processing result of sensor data as a WoT logical sensor. The logical sensor runs as a WoT entity. Applications can access the results of the processing by URIs via HTTP/HTTPs as if they were accessing an independent WoT sensor.
- Application developers define the flow of the sensor data processing as a pipeline of multiple processes. Once a data flow is executed, every result of the processes on the pipeline of the data flow becomes a WoT logical sensor.
- If a subset of the pipeline processes specified by an application already exists as a WoT logical sensor, the application replaces the subset of the pipeline to the existing WoT logical sensor. Thus, the application can reuse the existing WoT logical sensors, if available. The application program

developers do not need to concern about such pipeline reuse.

3.2. Example Scenario: Object Detection

In the following, we show a typical example use-case scenario of the WoT logical sensor.

In this example, we assume a data flow processing application that detects objects in images captured by cameras, counts the number of people using the detection results and controls air-conditioning equipment according to the number of people. In this case, the pipeline can be represented as, for example, "camera → object detection → people count → air conditioning."

When another application with a data flow that includes the same microservice arrives, the corresponding WoT logical sensor is applied. For example, if an application with data flow: "camera → object detection → persons count → presence management" is started, since the above air conditioning application has already been executed, this presence management application reuses the common part of the pipelines (Figure 2).

We assume such a redundancy often occurs in the edge computing environment because the locally deployed sensors are utilized by personalized applications that have slightly different data flows invoked by mobile users. If a microservice on the pipeline requires a kind of CPU heavy computations (e.g., object detection process), it is hard to accommodate multiple applications on a single edge computer. By using the WoT logical sensor, the processes of the microservices can be shared among multiple data flow applications. Thus, the required computation resources can be reduced.

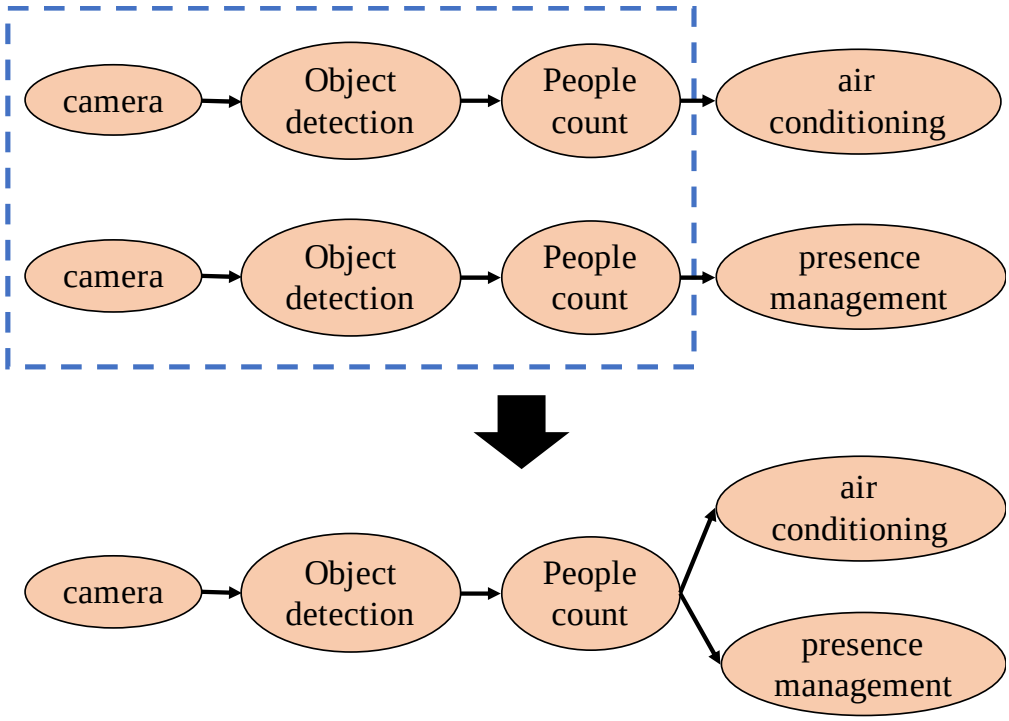

Figure 2. Example Scenario: Object detection

3.3. Data Flow Transforming in WLSA

In WLSA, we assume to place a repository on each edge network so that the platform can determine whether a part of the pipeline is already running in the edge network or not. The repository manages the running WoT logical sensors in the edge network. The repository enables applications to find the running WoT logical sensors by specifying a set of URIs that corresponds to a pipeline.

We denote a pipeline data flow D as follows:

$$D = \{d_0, d_1, \ldots, d_{|D|-1}\}$$

where d_i represents the i-th URI of the microservice. The order D corresponds to the order of the microservices to apply data as a pipeline. Note that we do not treat conditional branches in the current model. The data flow needs to be split into multiple sequential pipelines if there is a branch.

In WLSA, we propose a "data flow transforming (DFT)" to provide the logical sensor feature. Algorithm 1 shows how the data flow structure is transformed in WLSA.

The algorithm shows the behavior of the platform receives a request from an application to execute a data flow D. First, the entry that has a key with the longest common prefix (LCP) with D is retrieved from the repository (line 2). If no entry has a common prefix with D, then all subsets of D are treated as keys. In the algorithm, S corresponds to the key (line 6). The algorithm then starts an HTTP process p that responds to the result of the corresponding result to the GET request (line 7). Then, an entry with key S and value p is registered to the repository (line 8).

If there is an entry that has a common prefix with D, the corresponding process p is reused for D. That is, the common prefix (i.e., k) is eliminated from D and connected to p (line 12). After the above DFT process, the resulting data flow D' is started (line 13).

Algorithm 1: DFT Algorithm

```
1  upon receiving ⟨D⟩
2  k ← search LCP of D from repository;
3  if k = φ then
4      S ← φ;
5      foreach d_i ∈ D do
6          S ← S ∪ {d_i};
7          p ← start HTTP process to get result of d_i;
8          register S, p to the repository;
9      D' ← D;
10 else
11     p ← corresponding process for k;
12     D' ← {p} ∪ {D − k};
13 start D';
```

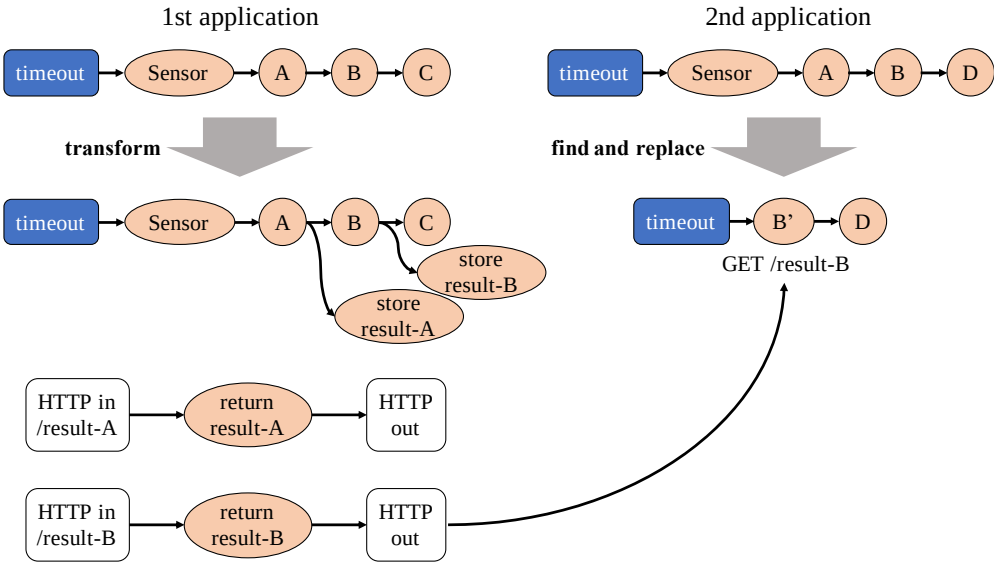

Figure 3. An example of DFT

Figure 3 shows an example of DFT. In this example, the 1st application periodically gets data from Sensor and processes A, B are applied as the data flow pipeline. In this example, C corresponds to the output of the data flow. Thus, there is no output for C. By the DFT, the outputs of processes A and B are stored on memories, and the HTTP processes that correspond to obtain these outputs are generated. When the 2nd application, in this case, the pipeline with A, B, and D, arrives, then the pipeline before B is replaced with B', which corresponds to the HTTP process that returns the result of B in the 1st application.

By the above algorithm, the platform automatically reuses the already running process that generates the same data processing result. DFT has an overhead to generate new HTTP processes for each subset of the pipeline. However, such a function does not require much computation or communication resources compared to starting new duplicated processes.

To share the processing results among multiple applications in the practical systems, data access rights must be considered in WLSA. It is necessary to control data access rights such as allowing the connection of the data flow pipeline only when the access right exists. Such a data access right management is out of the scope of this paper.

3.4. An Implementation using Node-RED

We are implementing a prototype of the proposed architecture using Node-RED [3] data flow processing framework. We apply MongoDB [10] as a repository implementation. The structure of the prototype system is shown in Figure 4.

The application requests a data flow to execute to the client computer (Desktop). The client computer has a front end to accept data flow requests.

The data flow is defined and launched by Node-RED on the client. When a new application on the Node-RED starts up, the DFT algorithm described in the former section, using MongoDB as a repository, is executed.

In our prototype implementation, the transformed data flow is offloaded to the Edge Server and executed by the Node-RED process on it. If there is no existing data flow in the repository, Node-RED on the edge server starts to use a microservice. The microservice is offloaded from cloud to the Edge Server according to the edge computing paradigm.

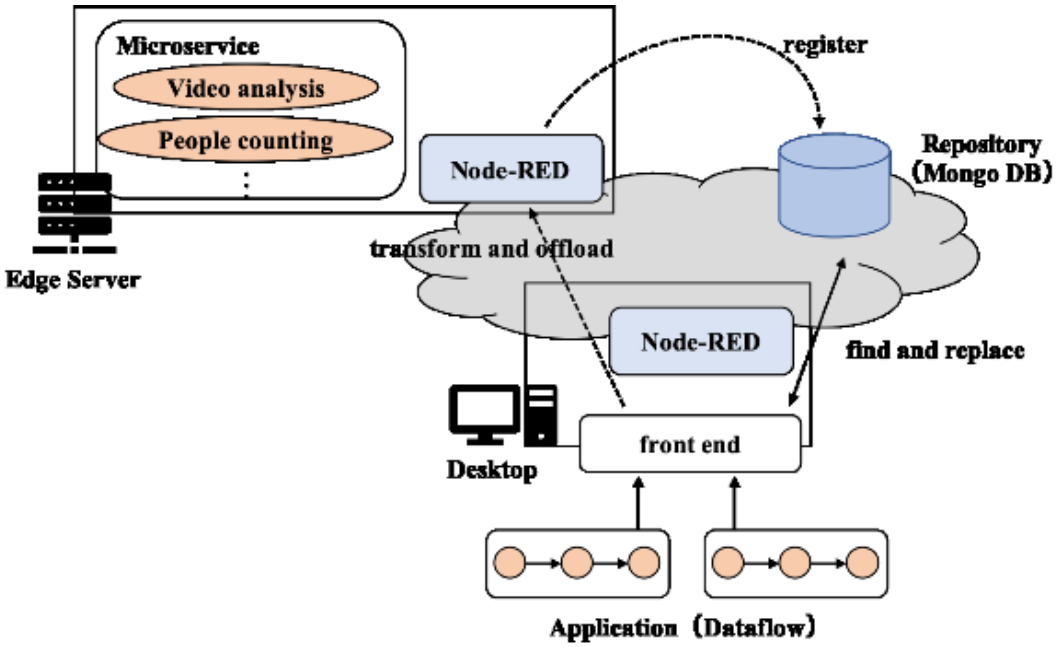

Figure 4. Configuration of the implemented system

4. Evaluation

4.1. Evaluation Setup

To show the effectiveness of the proposed WLSA, we conducted an evaluation using the prototype implementation described in the previous section.

We used Mac mini (Intel Core i6, 32GB memory) for the edge server and Macbook Pro (Intel Core i4, 8GB memory) for the client computer.

We measured communication delay, CPU usage, and memory usage for the evaluation. As a typical application, we used a data flow of "camera → object detection → people count". The "camera", "object detection", and "people count" processes are implemented as microservices. The camera is implemented as a WoT image sensor. For the evaluation, the camera runs as a dummy that generates a JPEG image, which includes a person in a room for all requests. The object detection application is implemented by YOLOv3 [11]. The parameters of the evaluation are summarized in Table 1.

TABLE 1. EVALUATION SETUP SUMMARY

Parameters	Setup (Value)
Edge Server	Mac-mini (Intel Core i6, 32GB memory)
Client	Macbook Pro (Intel Core i3, 8GB memory)
Data Generation Interval	3 (s)
Data size	27K bytes
Network bandwidth	2.4 Gbps WiFi
Object detection	YOLOv3
Number of tests	10
Number of applications	1 to 20

We compared WLSA with the following two methods in the same evaluation setup.

- Share a process on the edge (denoted as "SP")

  This case is a naive setup. Only one object detection microservice and one people count microservice are executed on the Edge Server. These processes are shared among all applications. The number of executed data flows increase as the number of applications grows in SP.

- Run multiple processes at the edge (denoted as "MP")

  This case corresponds to a "resource slicing" setup. Each application allocated its own edge computing resource (i.e., CPU core). In our environment, dedicated Node-RED process runs for each application.

4.2. Results

Figure 5 shows the processing latencies for each method. Note that the y-axis is shown by a logarithmic scale. The latencies are the average of 10 test results. In the case of SP and MP, the latency increased as the number of applications grown. The processing latency became very large when the number of applications was 6 for SP and 13 for MP. MP could not accommodate more number of applications because microservices no longer respond to requests due to network congestion. In the case of WLSA, the latency was less than 100ms in any number of applications. This is because, in WLSA, the HTTP process only needs to respond to the requests from applications by returning the stored result. Therefore, the computation load of the object detection was not increased as the number of applications grown. By this result, we could observe that the overhead of the WLSA to generate an extra HTTP process was small.

To see a more detailed reason why the latency was increased in SP and MP, we measured the CPU usage and memory usage of the Edge Server by changing the number of applications. Figure 6 and Figure

7 show the CPU usage and memory usage, respectively.

In SP, as the number of applications increased, CPU utilization increased to 15%, but memory usage did not change. This is because the processes only occupy one CPU Core. The saturation occurs because of CPU was exhausted by the object detection process. The request via network was locked until the previous object detection processing request was finished.

In MP, as the number of applications increased, CPU usage increased to more than 90%, and memory usage reached 32G (Max memory usage). This is because as the number of applications increased, the occupied CPU core increased. The max number of executed applications was 12 because the Edge Server has 12 CPU cores. We could see a saturation in the CPU utilization when the number of applications was 12 in Figure 6. That is, the CPU and memory were the bottlenecks in this environment.

Compared to these methods, WLSA was scalable for the number of applications. The CPU usage was kept around 4%, and memory usage did not increase as the number of applications grown. This is because a new microservice was not created even if the number of applications increased by reusing the processing result.

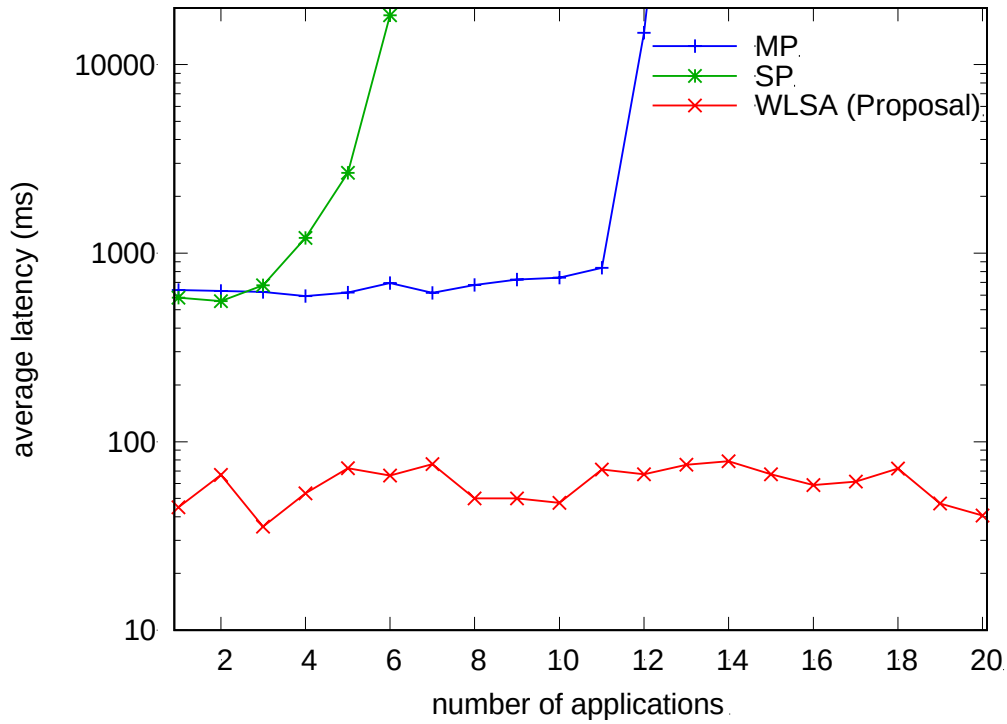

Figure 5. Process latency

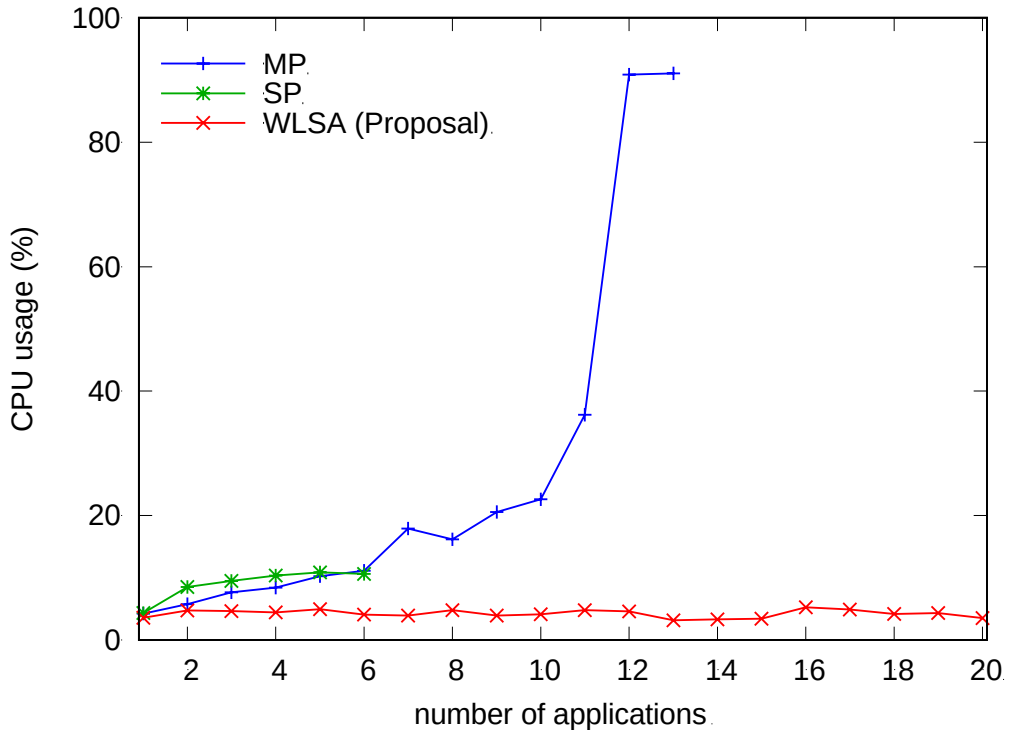

Figure 6. CPU usage

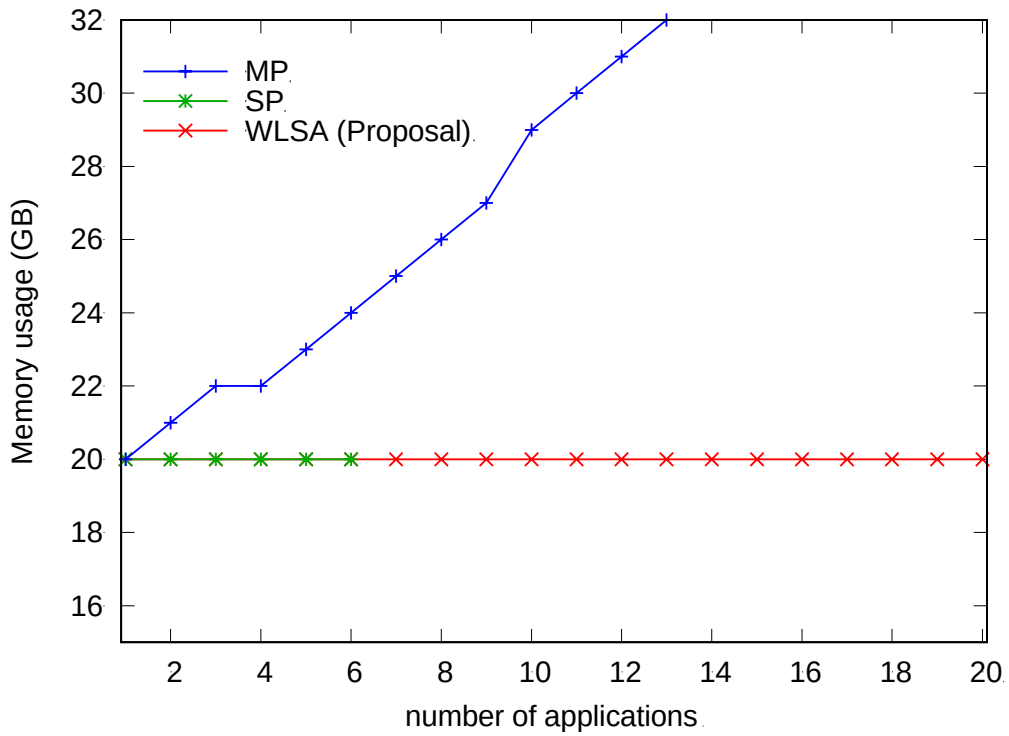

Figure 7. Memory usage

5. Related Work

There have been several types of research and systems for service discovery to reuse the existing IoT/WoT services. Mayer et al. [12] propose DiscoWoT, a semantic discovery service for Web-enabled smart things. The service is based on the application of multiple Discovery Strategies to a Web resource's representation, where arbitrary users can create and update strategies at runtime using DiscoWoT's RESTful interface. Kallab et al. [13] proposes a scalable service discovery method for the dynamic composition of RESTful services including WoT. These researches aim to provide a function for the service discovery. Therefore, these technologies can be replacements for the repository in WLSA.

Mrissa et al. [14] propose a model that supports semantic discovery and invocation of functionalities for the WoT. Based on an architecture called Avatar, they show a model to describe the capabilities of objects and enable reasoning about these capabilities to expose high-level functionalities into WoT applications. Avatars rely on Web languages, protocols, and reason about semantic annotations to dynamically drive connected objects, exploit their capabilities, and expose user-understandable functionalities as Web services. Though the Avatar architecture provides a dynamic invocation feature of WoT, it does not reuse nor share the processing results, which is a key feature in WLSA.

Several existing studies have treated data flow platforms in clouds and edge computing networks. FRED [15] is a Node-RED platform on cloud that hosts data flow mashup tools for multiple IoT applications. It provides multiple Node-RED runtimes on a server instance on the cloud. The architecture of FRED corresponds to the comparison method "MP" in our evaluation.

The article [16] proposes a data flow processing platform that provides an offloading and auto-scaling feature in an edge computing environment using a locality-aware structured overlay network. Ishihara et al. [17] propose an effective deployment method of data flow processing components in the hierarchical edge computing network. However, none of the existing studies have proposed the logical sensor approach that treats the processing results as sensors.

About the logical sensors, there have been several cloudbased services. Xively [18] (formerly known as Pachube) is one of the earliest online services which allows connecting sensors to the web. Xively has its own web site for managing the sensor data and sensor access APIs. ThingSpeak [19] provides open APIs to store and retrieve data from device assets or things via LAN or using HTTP over the Internet. There have also been some studies about the logical sensors in the literature. Misra et al. [20] proposed a method of dynamic mapping of virtual sensors in sensor-cloud for provisioning high quality of Sensors-as-a-Service (Se-aaS) in the presence of multiple sensor-owners and heterogeneous sensor nodes. The article [21] and related articles proposed a social sensor architecture that provides features to search for similar data from social network services and the processing results. Some cloud-based virtual sensor platforms are already industrial services. SensorCorpus [22] provides a "sensor data stream processing" feature that combines several analysis modules into real-time processing to provide logical sensors. However, the application developers of the logical sensors in the above existing studies or services need to define and deploy logical

sensors independently, which is difficult in the edge computing environment where the applications are deployed and executed dynamically. In our WLSA architecture, it is not necessary for the application developers to conscious about reuse of the data flow definition in such a dynamic environment, which differs from these existing approaches.

6. Conclusion

In this paper, we propose a WLSA that can reduce the resources required for data processing by reusing the processing result as a virtual WoT sensor. We aimed to solve the processing delay caused by the increase of WoT application caused by the power-saving device of the sensor device by WLSA. It is shown that the use of WLSA can reduce CPU usage and memory usage, and the processing time can also be shortened.

As the sensor data processing, there would be a required deadline to complete a sensor data process. While WLSA could achieve small processing latency, it does not treat such deadlines. Extending WLSA considering such a deadline of the sensor data processing is our future work.

Furthermore, in this paper, if there is similar data flow, the microservices are all executed at the edge. Considering the edge and cloud computing resources (CPU, memory), dynamically distributing the microservices that make up the data flow to both the edge and the cloud is also our future work. The security considerations appropriate for edge computing such as the approach of zero trust networking [23] should be taken into account.

More general logical sensor definitions, more complete and efficient implementations, and applying to the real usecase applications are also our future work.

References

[1] D. Guinard, V. M. Trifa, and E. Wilde, “Architecting a mashable open world wide web of things,” ETH Zurich, Tech. Rep., Feb. 2010.

[2] “Webthings gateway.” [Online]. Available: https://iot.mozilla.org/gateway/

[3] M. Blackstock and R. Lea, “Toward a Distributed Data Flow Platform for the Web of Things (Distributed Node-RED),” in Proceedings of the 5th International Workshop on Web of Things. Association for Computing Machinery, 2014, p. 34–39.

[4] H. Tanaka, M. Yoshida, K. Mori, and N. Takahashi, “Multi-access edge computing: A survey,” Journal of Information Processing, vol. 26, pp. 87–97, 2018.

[5] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, “Mobile edge computing—A key technology towards 5G,” ETSI white paper, vol. 11, no. 11, pp. 1–16, 2015.

[6] R. Sharma and A. Singh, “Istio gateway,” in Getting Started with Istio Service Mesh. Springer, 2020, pp. 169–192.

[7] “Migration to Object Technology,” 1995.

[8] M. Zaharia et al., “Apache Spark: A Unified Engine for Big Data Processing,” Commun. ACM, vol. 59, no. 11, p. 56–65, 2016.

[9] “Apache beam.” [Online]. Available: https://beam.apache.org/

[10] K. Banker, MongoDB in Action. USA: Manning Publications Co., 2011.

[11] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” arXiv, vol. 1804, 2018.

[12] S. Mayer and D. Guinard, “An extensible discovery service for smart things,” in Proceedings of the Second International Workshop on Web of Things, 2011, pp. 1–6.

[13] L. Kallab, R. Chbeir, and M. Mrissa, “Automatic K-Resources Discovery for Hybrid Web Connected Environments,” in 2019 IEEE International Conference on Web Services (ICWS). IEEE, 2019, pp. 146–153.

[14] M. Mrissa, L. M´edini, and J.-P. Jamont, “Semantic discovery and invocation of functionalities for the web of things,” in 2014 IEEE 23rd International WETICE Conference. IEEE, 2014, pp. 281–286.

[15] M. Blackstock and R. Lea, “Fred: A hosted data flow platform for the iot,” in Proceedings of the 1st International Workshop on Mashups of Things and APIs, 2016, pp. 1–5.

[16] Y. Teranishi, T. Kimata, H. Yamanaka, E. Kawai, and H. Harai, “Dynamic Data Flow Processing in Edge Computing Environments,” in Proc. of COMPSAC 2017, 2017, pp. 935–944.

[17] S. Ishihara, S. Tanita, and T. Akiyama, “A dataflow application deployment strategy for hierarchical networks,” in 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), vol. 1. IEEE, 2019, pp. 326–335.

[18] “Xively.” [Online]. Available: https://xively.com/

[19] “Thingspeak.” [Online]. Available: https://thingspeak.com/

[20] S. Misra and A. Chakraborty, “QoS-Aware Dispersed Dynamic Mapping of Virtual Sensors in Sensor-Cloud,” IEEE Transactions on Services Computing, 2019.

[21] Z. Aiko, K. Nakashima, T. Yoshihisa, and T. Hara, “A social sensor visualization system for a platform to generate and share social sensor data,” in 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), vol. 2. IEEE, 2018, pp. 628–633.

[22] “Sensorcorpus.” [Online]. Available: https://sensorcorpus.com/

[23] E. Gilman and D. Barth, Zero Trust Networks. O’Reilly Media, Incorporated, 2017.